**创业者** [**郭凛**](http://www.zhihu.com/people/guo-lin)**：如果纯以编码能力来看，可分为五类**

1. 拷贝型

对实现功能几乎没有思路，只会改代码，却不会写代码。对编程毫无兴趣，只是以此糊口而已。

2. 新手型

经验有限，程序员并不完全知道要如何实现功能，需要通过学习、寻找资料等方式来解决问题。这很难有多余的心思去考虑边界条件、性能、可读性、可扩展性、编码规范等问题，因此代码bug较多，稳定性不高，常常会发生开发花费1个月，改bug却要改上好几个月的事情。

3. 学习型

这类程序员对所在领域的语言已经比较了解，对于一般功能可以有较为清晰的实现思路，给出需求时可以通过自己的思路来实现，并且会一定程度上考虑边界条件和性能问题。但仅此而已，他们对可读性和可扩展性考虑很少，也没有项目级别的考虑，主要是希望通过实现代码来练手或是学习。

**这类程序员最大的表现在于喜欢“创造代码”**，即使有现成的实现，他们也希望自己来实现一套，以达到“学习”的目的。他们不喜欢复用别人的代码，看见项目中别人实现了相类似的功能，他们会以“需求不同”的借口来自己重新实现一套。这类人一般来说对技术有着较为浓厚的兴趣，希望能够通过项目来进行学习。从项目的角度来说，这种做法最大的麻烦在于开发周期可能较长（相比直接使用现成的实现），并且会使得项目代码膨胀，影响未来的维护。但这类程序员由于有兴趣，如果好好培养或许会成为明天的牛人。

4. 实现型

这类程序员一般有较为丰富的经验，由于写得太多，因此不再追求“创造代码”来进行学习，同时对所在领域可能涉及的很多第三方框架或是工具都比较熟悉，当接受到产品需求时，对功能实现方案已经了然于胸，因此他们可以快速的实现需求，并且对边界、性能都有一定程度的考虑。因为能够快速实现需求功能，经常会被团队评价为“牛人”。但他们一般仅仅停留在“完成功能”级别上，对代码的可读性、可扩展性、编码规范等考虑较少，对项目总体把握也较少（例如控制项目膨胀、方便部署等架构级别的东西）。

**这类程序员最大的表现在于喜欢“开发项目”，却不喜欢“维护项目”。**他们产出的代码最大的问题就是维护较为困难，可能过上几个月回头看自己的代码都会晕头转向。因此即使是自己写的代码，仍然不愿意维护，一般会苦了后来人。因为接口设计的缺乏，当需求变更时，发现代码要改的东西太多，然后抱怨需求变化，却很少认为是自己的代码问题。这样的项目如果经过长时间的变更维护，最终会变得难以维护（一般表现在需求变更响应时间越来越长）甚至无法维护，最终要么是半死不活，要么是被推倒重来。

5. 架构型

这类程序员比实现型更进一步，他们经验丰富，对相关框架和工具等都很熟悉，“完成功能”、“稳定性”、“性能”这些已经不再是他们的追求，更优美的代码、更合理的架构才是目标。

这类程序员代码设计大多建立在对需求的详细了解和对需求变更的预测上——可扩展性较好；代码细节也尽量多的考虑边界情况、性能——稳定高效；代码命名和注释都恰到好处——可读性较高；同时在开发过程中他们会不断重构，对代码做减法——保证项目可持续发展等等，但由于考虑问题较多，单从“实现功能”阶段来看，完成速度不一定会比“实现型”要快。只是到了项目中后期优势才会慢慢体现出来。

**知乎删代码工程师**[**安江泽**](http://www.zhihu.com/people/gnap)**：创造型、管家型、学院派、钟表型、折腾型**

1. 创造型/创新型

以创造/创新为乐的程序员。以新的方法解决现有问题，发明方法解决尚未解决的问题。问题本身并不限于技术领域。硅谷很多技术创始人具备这类的品质。虽然他们创业后都不再全职写代码，但他们这种品质贡献了他们事业的第一步。

2. 管家型

以管系统演化为乐的程序员。有洁癖，有强迫症，容不得系统有一点的瑕疵。如果你招聘了一个程序员/架构师，2-3年后你的互联网产品再也没有在可见/可扩展性上遇到瓶颈，他多半是这种程序员。

3. 学院派型

通常在CS某一领域拥有博士学位，或者深受他们的影响的人。对该该领域兴趣浓厚，通常参与企业核心竞争力相关的研发工作。当然，也有一部分也具备第一种特质，中途去改变世界去了。

4. 经验型

在行内具有多年丰富的经验，可承担责任，考虑全面。团队拥有了这种人，通常会少走很多弯路。但面对新的挑战还是要和其他人站在同一起跑线上的。

5. 钟表匠型

喜欢并擅长把一个系统从里到外研究通透的人。分享出来都是从里到外让人垂涎欲滴的干货。很多后端工程师都是这种类型。

6. 折腾型

擅长“折腾”，方案没有最好的，只有最难的。兴趣广泛动手能力强，通过折腾留下很多技术积累。想用哪一块的新技术？先问问他把，8成他很久前就已经玩过了。

7. 肉盾型

哥甲高血条长，加班如同切菜一般，逼急了不睡觉也行。只伸手要过两件东西：原型和咖啡。适合打闪电战。当然，可能事后要打打后勤重构下代码。

**iApp4Me创始人、iOS开发者**[**郝培强**](http://www.zhihu.com/people/tinyfool)**：七大标准 逻辑思维很重要**

优秀程序员的标准：

1. 要懂逻辑，这点其实挺难达到，很多很不错的程序员其实逻辑能力不行，所以我觉得限制了他们更深一层次的发展，或者说限制了他们解决特别复杂问题的能力。
2. 要有分析能力。这也是解决复杂问题不可或缺的能力。
3. 要会自学，技术发展速度很快，如果没有自学能力会被淘汰的。所以在中国英语要好，不然看不懂最新的技术走向。
4. 要有良好的时间管理习惯和意识。
5. 要有耐心，肯于付出。
6. 要有良好的工作习惯。
7. 会看且勤于看文档。

**产品管理、IT项目管理**[**何明璐**](http://www.zhihu.com/people/cmmi)**：优秀程序员具有较明显的自恋和偏执特征**

1. 对编程有浓厚的兴趣，由兴趣能驱动自己的探索和钻研精神，优秀程序员一般喜欢用业余时间探索和自己琢磨点东西，研究点开源框架和源代码，关注新技术，喜欢研究和探索算法，逻辑方面的内容。
2. 优秀程序员一般比较懒于做重复的事情，对于重复事情有独有的程序员思维，即通过程序或其它自动化方式解决重复问题，而留更多时间去思考。
3. 优秀程序员更多的会从简单功能实现，花哨的界面转移到对安全，性能，可扩展等底层架构的关注。
4. 优秀程序员一般有较明显的自恋和偏执特征，有较为希望独自工作的特征，比较难于被非技术背景管理者驾驭，高度强调自我能力价值的被认可，而且很多优秀程序员并不是好教练。
5. 优秀程序员一般有较为扎实的算法，数据结构等理论知识的基础。这些让他们在迅速进入新领域的时候快人一筹。

**总结：**

不管是优秀程序员还是一般程序员，小编认为只要你喜欢思考，爱学习，对编码有兴趣，那么你就是最好的。当然掌握技术才是硬道理，套用别人的一句话“不想当CEO的程序员，不是好的产品经理”。程序员们，加油吧！

**0. 掌握基础**

　　对于任何行业任何工作来说，融会贯通是获得成功的关键。一个人想要成为优秀程序员，就必须有坚实的基础。核心理念的理解会帮助你用最好的方法设计和实施出最完美的方案。如果你感觉到不能掌握核心的计算机科学或者某个编程语言的知识点，现在开始回顾基础一点都不晚。

**1. 开始为你写过的代码贴上标签(怎么做，做什么)**

　　我发现优秀的程序员和普通程序员之间有一条明显的分界线，优秀程序员渴望深究每行代码做了些什么，以及如何实现的。有那么一小群人，他们一定要弄懂每一行代码。我知道如果时间非常紧张的时候，我们经常不能够遵循这个原则，仅仅写出些代码，知道它实现了需要的功能。如何处理这种情况又是另外一个话题，但是作为一个程序员，我们经常要尽可能深入探索。相信我，随着时间的推移这会变成一个习惯，之后每次你都会不知不觉的这样做了。

**2. 帮助他人，你可以学到更多**

　　许多人都有个共同特点，只有当他需要帮助的时候，他才会求助于论坛或者群。优秀程序员不同之处在于他们会经常浏览论坛去帮助他人。相比较于靠别人帮助解决问题，他们帮助他人让自己学到更多。在一个团队中也是一样，帮助他人解决问题收获更多。相信我，了解他人的问题，思考并最终提供解决方案吧，你会比之前学到的更多。

**3. 写简单易懂，有逻辑性的代码**

　　正如KISS原则(保持短小精悍)适用于生活的方方面面一样，它也适用于编程。尽量写有逻辑的代码，避免复杂化。有时人们写复杂的代码仅仅是为了展示他们有能力写出这样的代码。我的经验告诉我，简单而富有逻辑的代码非常的好，会产生的问题更少，也更容易扩展。我记得一句话：

　　好的代码本身就是最好的文档。当你想要加注释的时候，问问你自己，“我能如何改进我的代码，才能不写注释也读得懂呢?”

**4. 花更多的时候分析问题，你将会花更少时间去解决问题**

　　花更多的时间理解和分析问题，然后再设计方案吧。你会发现剩下的事情很容易了。设计不是说要用建模语言和工具，可以是仅仅看看天空在脑子里构思。那些在遇到问题就开始敲代码的人往往会最终偏离需求。

　　如果你洗澡的时候不能理清整个程序的结构，那么你还没准备好开始编程。 – Richard Pattis

**5. 成为第一个检查你的代码的人**

　　虽然有一点点难，但试着在其他人修改你代码之前修改它，随着时间的推移，你会写出几乎没有bug的代码。对你的代码做没有任何偏见的检查，也不要犹豫让其他人来检查你的代码。和其他优秀的程序员一起工作，接受他们的意见能够帮助你也成长为一个优秀的程序员。

**6. 不要迷失在快速更迭的科技世界**

　　在IT行业中，我经常遇见许多人，他们对现在的工作不满，甚至离开它去追寻新的工作，理由是因为他们想要学习最新的科技。我们每天听到的都是新工具、接口、框架，能让程序更简单，速度更快。这在科技世界中司空见惯并会一直如此。但是最基本的最核心的科技变化比那些框架、工具和接口的变化小得多。就像大海，大海表面上波涛翻滚，但大海深处却相当平静，而那里才是水生生物生存的地方。所以将你自己置身于核心技术的大海深处吧。举个例子，在Java企业级应用中，每个星期都会出现新的框架，但是核心的技术是不变的，譬如基于客户端-服务器端的请求，MVS模式，filters/servlets/JSP，数据源绑定，XML解析等等。所以要花功夫去学习核心概念，而不是去担忧日新月异的框架和工具的出现。相信我，有了核心技术的基础，你会发现学习新的框架，工具以及接口变得更容易了。

**7. 应急方案不会持续很长时间**

　　很多时候软件师们会采用应急方案(可能是时间不够，可能是问题理解不透彻或者是缺少经验)。但是这样往往带来的结果是，代码混乱，缺乏可扩展性和维护性，后来浪费了更多的时间。请在你清楚整个方案后再开始实施。我知道有些情况下应急方案无可避免，就好像，一个人应该说实话，但是你在某些情况下不得不说谎。

**8. 阅读文档**

　　阅读很多文档是作为优秀程序员的必备习惯之一。可能是产品说明书、JSR、API文档、教程等。阅读文档帮助你获得必要的基础知识，写出更好的代码。

**9. 你可以学习他人的代码**

　　我接触过一些优秀的程序员，他们在IDE里面一直放着JAVA源代码，并经常在日常工作中阅读和查阅。他们这样做不仅是满足求知的渴望，还从中学习了如何写优秀的程序。阅读和查阅那些你认为可靠的出名的开源代码或者比你资深的程序员的代码，你会写出更棒的程序。(扩展阅读：《阅读优秀代码是提高开发人员修为的一种捷径》)

**最后，我想说的但没有将其列在上面的：不要和他人比较**

　　和他人的比较往往会造成消极的情绪和不健康的竞争。每个人都有他的长处和短处。扬长避短很重要。我也经常看到一些基础很厉害的程序员范很低级的错误。所以仅仅和自己比较，列举你自己需要改进的地方，并持续改进它。编程是很有意思的事情，享受它吧。

　　任何一个傻瓜都能写出计算机能理解的程序，而优秀的程序员却能写出别人能读得懂的程序。